Applications of automatic differentiation in topology optimization

Nørgaard, Sebastian A.; Sagebaum, Max; Gauger, Nicolas R.; Lazarov, Boyan Stefanov

Published in:
Structural and Multidisciplinary Optimization

Link to article, DOI:
10.1007/s00158-017-1708-2

Publication date:
2017

Document Version
Peer reviewed version

Noname manuscript No.
(will be inserted by the editor)

Applications of automatic differentiation in topology optimization

Sebastian A. Nørgaard · Max Sagebaum · Nicolas R. Gauger · Boyan S. Lazarov

Received: date / Accepted: date

Abstract The goal of this article is to demonstrate the applicability and to discuss the advantages and disadvantages of automatic differentiation in topology optimization. The technique makes it possible to wholly or partially automate the evaluation of derivatives for optimization problems and is demonstrated on two separate, previously published types of problems in topology optimization. Two separate software packages for automatic differentiation, CoDiPack and Tapenade are considered, and their performance and usability tradeoffs are discussed and compared to a hand coded adjoint gradient evaluation process. Finally, the resulting optimization framework is verified by applying it to a non-trivial unsteady flow topology optimization problem.

Keywords Topology optimization · Automatic differentiation · Lattice Boltzmann

1 Introduction

Automatic differentiation, also at times called algorithmic differentiation, is a technique that, according to Griewank and Walther (2008) "has been rediscovered and implemented many times, yet its application still has not reached its full potential". Automatic differentiation (AD) allows for the exact evaluation of the Jacobian of an arbitrarily complicated differentiable function, by partitioning the function into a sequence of simple operations, which are by themselves trivially differentiable. This process can be automated by software, allowing developers to focus on the solution of the problems requiring differentiation, rather than the derivation and implementation of code for evaluating derivatives. This potential for easily evaluated derivatives makes AD very useful for design optimization, especially for highly non-linear problems (Albring et al 2016; Nemili et al 2014; Zhou et al 2017; Özkaya et al 2016). Despite this, to the authors knowledge, there have been only few applications of AD for density based topology optimization—the only example the authors are aware of is the paper by Łaniewski Wołłk and Rokicki (2016). Thus, the aim of the presentation is to discuss the application details and to demonstrate AD for two topology optimization problems in computational mechanics.

An extensive review of topology optimization itself is beyond the scope of this paper, but the interested reader is referred to the monograph by Bendsøe and Sigmund (2004), as well as the more recent review paper by Sigmund and Maute (2013).

1.1 Automatic differentiation

The goal of this section is to give a brief introduction to AD. For an extensive and more general treatment, the reader is referred to the introductory text by Griewank and Walther (2008). To simplify the discussion, assume a continuous function $F : \mathbb{R}^n \to \mathbb{R}^m$, with the Jaco-

Sebastian A. Nørgaard
Department of Mechanical Engineering, Technical University of Denmark, DK-2800 Lyngby, Denmark
E-mail: sebnorg@mek.dtu.dk

Max Sagebaum
TU Kaiserslautern, Chair for Scientific Computing, 67663 Kaiserslautern, Germany

Nicolas R. Gauger
TU Kaiserslautern, Chair for Scientific Computing, 67663 Kaiserslautern, Germany

Boyan S. Lazarov Department of Mechanical Engineering, Technical University of Denmark, DK-2800 Lyngby, Denmark

bian matrix $F' : \mathbb{R}^n \to \mathbb{R}^{m\times n}$. Further assume that a routine (i.e. a particular computer implementation) exists to evaluate F. In the AD literature, F is often called the *primal function*. Even though F may be arbitrarily complicated, its concrete implementation may be decomposed into a series of simple operations (e.g. additions, multiplication, elementary functions such as the trigonometric functions) which are individually easy to differentiate exactly. The differentiated value of each operation can then be propagated to the next by the chain rule. This idea of propagation can be applied in two ways, either starting from the input vector $x \in \mathbb{R}^n$, which results in the *forward mode*, or from the output vector $y \in \mathbb{R}^m$, which results in the *reverse mode*. Since the full mathematical details of AD are beyond the scope of the paper, each mode will be demonstrated by means of a very simple example.

For the forward mode, consider the function

$$\begin{aligned} &f(x) : \mathbb{R} \to \mathbb{R}^2, \\ &y_1 = \cos(\cos(x)), \\ &y_2 = \exp(y_1). \end{aligned} \tag{1}$$

An implementation of (1) might evaluate the function like so:

$$\begin{aligned} &v_1 = \cos(x), \\ &v_2 = y_1 = \cos(v_1), \\ &v_3 = y_2 = \exp(v_2), \end{aligned} \tag{2}$$

where the variables v_i can be considered intermediate values or "computational steps" taken to evaluate the function. Using these steps, the derivative of f with respect to x can be obtained as:

$$\begin{aligned} &\dot{v}_1 = -\sin(x), \\ &\dot{v}_2 = \dot{y}_1 = -\sin(v_1)\dot{v}_1, \\ &\dot{v}_3 = \dot{y}_2 = \exp(v_2)\dot{v}_2, \end{aligned} \tag{3}$$

where the dot denotes differentiation with respect to x. In general, the forward mode allows the evaluation of the expression:

$$\dot{y} = F'(x)\dot{x}, \tag{4}$$

where $\dot{x} \in \mathbb{R}^{n\times 1}$ is called the *seed direction*.

For the reverse mode, consider the function:

$$\begin{aligned} &g(x_1, x_2) : \mathbb{R}^2 \to \mathbb{R}, \\ &y = \cos(\cos(x_1 x_2^2)), \end{aligned} \tag{5}$$

a possible evaluation procedure for this function is:

$$\begin{aligned} &v_1 = x_1, \\ &v_2 = x_2, \\ &v_3 = v_2^2, \\ &v_4 = \cos(v_1 v_3), \\ &v_5 = y = \cos(v_4). \end{aligned} \tag{6}$$

The *adjoint variables*, $\bar{v}_i = \partial y/\partial v_i$, may now be evaluated by stepping through the evaluation (6) in reverse order:

$$\begin{aligned} &\bar{v}_5 = 1, \\ &\bar{v}_4 = -\bar{v}_5 \sin(v_4), \\ &\bar{v}_3 = -\bar{v}_4 \sin(v_1 v_3) v_1, \\ &\bar{v}_2 = \partial g/\partial x_2 = 2\bar{v}_3 v_2, \\ &\bar{v}_1 = \partial g/\partial x_1 = -\bar{v}_4 \sin(v_1 v_3) v_3. \end{aligned} \tag{7}$$

In general, the reverse mode evaluates the expression:

$$\bar{x}^T = \bar{y}^T F'(x), \tag{8}$$

where $\bar{y} \in \mathbb{R}^{m\times 1}$ is termed the *weight functional*.

Note that the two examples given above are intentionally simplistic, as they serve only to demonstrate the principle of AD at the most basic level. Evaluating more sophisticated functions, one has to consider issues such as branching, potential instabilities caused by differentiation close to singularities or discontinuities, and the influence of round-off errors on the final result. Dealing with these things is an active area of research which is beyond the scope of this paper. The authors simply note that none of the examples shown in the following sections exhibit pathological behavior, and that the gradients obtained with AD in all cases have been verified by a finite difference check.

The expressions (4) and (8) above are general, but by choosing a standard basis seed direction or weight functional (e.g. $\dot{x} = \begin{bmatrix}1\ 0\ 0\ \dots\end{bmatrix}^T$), equation (4) and (8) allow the evaluation of a column or row of the Jacobian matrix, respectively. While both modes have similar mathematical properties, evaluating the reverse mode requires more memory since intermediate values and operations must be stored in order to step through them in reverse order. AD packages supporting the reverse mode generally provide a storage object—often called a *tape*—which is responsible for storing the information necessary to reverse the function F.

For topology optimization, the function of interest is typically the objective function, $F_{\text{objective}} : \mathbb{R}^{N_d} \to \mathbb{R}$, where N_d is the number of design variables. This makes the reverse mode the obvious choice, since the sensitivities

$$F'_{\text{objective}} = \begin{bmatrix} \frac{\mathrm{d}F'_{\text{objective}}}{\mathrm{d}s_1} & \frac{\mathrm{d}F'_{\text{objective}}}{\mathrm{d}s_2} & \dots \end{bmatrix},$$

can be computed with a single evaluation of the reverse mode (8), in the same manner that hand derived adjoints allow. It should be stressed that for topology optimization, N_d is typically much larger than in other structural optimization problems, since in size and shape optimization the geometry of the design is

represented by a much smaller number of parameters. In addition, note that the reverse mode is not fundamentally different from a hand derived discrete adjoint approach; its purpose is to reduce the burden of implementing the adjoint.

Generally, there are two approaches to implementing an AD package: source transformation and object overloading. Source transformation, as the name implies, provides a program which takes as input the source code to be differentiated, and outputs new source code which evaluates the derivative of the original source. An example of this type of implementation is Tapenade (Hascoët and Pascual 2013; Tapenade website 2016), which provides a convenient online server on which users can upload their source code, which will then get differentiated and served back. The advantage of this approach is that the differentiated code can be inspected directly, and if necessary, the user can manually optimize it to improve the execution speed of the application. Of course, if one chooses to do this, some convenience is sacrificed since the differentiation procedure is no longer fully automatic. Additionally, should the source code for the primal function change, the source transformation procedure—possibly including hand optimization—must be repeated.

The second approach, operator overloading, takes advantage of a feature of certain programming languages (notably C++ and Fortran 90) which allows the user to define basic operations such as addition and multiplication on user-defined types. This is exploited in AD libraries to provide types which perform both primal and differentiated computations. The function to be differentiated is then overloaded to accept these library types as input—rather than intrinsic floating point types such as `double` in C++. The resulting values can then be queried for their gradient as well as primal values. While this approach is typically slower than source transformation, it was shown by Hogan (2014) that in C++, expression templates could be used to achieve execution speeds which are competitive with source transformed code. The great advantage of this approach is the convenience. The code for evaluating the primal function can be reused without further implementation effort to evaluate the gradients. Furthermore, any modifications made to the primal code will be immediately reflected in the differentiated output, without requiring further involvement from the user. An example of this type of implementation is CoDiPack (CoDiPack website 2016). The CoDiPack library is header only, meaning that the code can simply be included in the application code to be differentiated, without any pre-compilation step. For further information, the interested reader is referred to the CoDiPack website cited above as well as Albring et al (2015a,b). The two packages presented above will be used to solve the optimization problems presented in this paper. For a much more complete list of AD packages, the interested reader is referred to the online list available at the AutoDiff website (2016).

The remainder of this paper is organized as follows: first AD is demonstrated for a relatively simple 1D wave propagation problem in Section 2. The example allows for easy comparison between hand written adjoint differentiation and fully automatic differentiation. The readers familiar with traditional adjoint analysis applied to transient topology optimization problems will identify immediately the similarities between the tape (the storage object in AD) and the storage of the forward solution in transient optimization problems. In the following Section 3 the applicability of AD is demonstrated for more complex optimization of transient fluid mechanics problems (Nørgaard et al 2016), where the explicit form of the Jacobian of the state equations including the boundary conditions is practically impossible to be derived by hand. The advantages and the disadvantages of AD are discussed and demonstrated in details, and finally the article is completed with a topology optimized example of a fluid device for oscillatory fluid input.

2 Application to transient wave propagation problems

The goal in this section is to demonstrate AD for well known one dimensional wave propagation problem (Dahl et al 2008; Lazarov et al 2011). The aim of the example is to compare and discuss the applicability of AD for complex topology optimization problems where significant amount of time is spend on derivation and implementation of sensitivities. The optimization problem is given as

$$\begin{aligned} \min_{s} :& J(\boldsymbol{s},\boldsymbol{u}) = \int_0^T z\left(\boldsymbol{s},\mathbf{u}\right) \mathrm{d}t, \\ \text{s.t.} :& \boldsymbol{r}\left(t,\boldsymbol{s},\mathbf{u},\dot{\mathbf{u}},\ddot{\mathbf{u}}\right) = 0, \quad t \in [0,T], \\ & g_i\left(\boldsymbol{s},\mathbf{u},\dot{\mathbf{u}},\ddot{\mathbf{u}}\right) \leq 0, \quad i \in \{1,\ldots,N_g\}, \\ & \boldsymbol{s} \in \mathcal{D}_{\mathrm{ad}}, \end{aligned} \tag{9}$$

where, $\boldsymbol{r}\left(t,\boldsymbol{s},\mathbf{u},\dot{\mathbf{u}},\ddot{\mathbf{u}}\right) = 0$ is the discrete form of the considered linear elastic state problem written in residual form, $\mathbf{u}$ is a vector with nodal displacements, $\dot{\mathbf{u}}$ is a vector with nodal velocities, $\ddot{\mathbf{u}}$ is a vector with nodal accelerations, $\boldsymbol{s}$ is the design vector with relative element densities, $J\left(\boldsymbol{s},\mathbf{u}\right)$ is the objective function and $g_i\left(\cdot\right), i \in \{1,\ldots,N_g\}$ is a set of additional constraints.

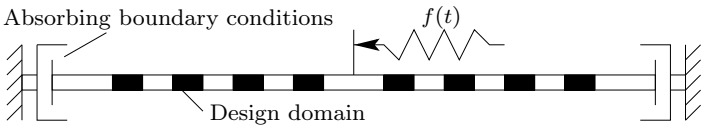


Fig. 1 Optimization setup. Absorbing boundary conditions are applied at both ends of the wave guide.

The residual form is given as

$$\boldsymbol{r}\left(t,\boldsymbol{s},\mathbf{u},\dot{\mathbf{u}},\ddot{\mathbf{u}}\right)=\mathbf{f}(t)-\left[\mathbf{M}\left(\boldsymbol{s}\right)\ddot{\mathbf{u}}+\mathbf{C}\left(\boldsymbol{s}\right)\dot{\mathbf{u}}+\mathbf{K}\left(\boldsymbol{s}\right)\mathbf{u}\right], \tag{10}$$

where the mass, damping and stiffness matrices $\mathbf{M}\left(\boldsymbol{s}\right)$, $\mathbf{C}\left(\boldsymbol{s}\right)$, and $\mathbf{K}\left(\boldsymbol{s}\right)$ are obtained by standard finite element assembly procedures. For every element the local matrices are obtained using linear interpolation between the matrices for two different materials, i.e,

$$\mathbf{M}\left(\boldsymbol{s}\right)_e=\left(1-s_e\right)\mathbf{M}_0+s_e\mathbf{M}_1, \tag{11}$$

$$\mathbf{C}\left(\boldsymbol{s}\right)_e=\left(1-s_e\right)\mathbf{C}_0+s_e\mathbf{C}_1, \tag{12}$$

$$\mathbf{K}\left(\boldsymbol{s}\right)_e=\left(1-s_e\right)\mathbf{K}_0+s_e\mathbf{K}_1. \tag{13}$$

An external excitation is applied as a time dependent nodal force in the middle of the computational domain

$$f(t)=\begin{cases}\cos(2\pi f_c(t-t_0))e^{-\delta(\frac{t}{t_0}-1)^2}, & t\geq 0,\\ 0, & t<0,\end{cases} \tag{14}$$

where t_0 is the center of the wave packet in the time domain, f_c is the central frequency, and δ defines the bandwidth (Dahl et al 2008). The excitation generates two Gaussian wave packets propagating towards the two ends of the wave guide. The set up is shown in Fig. 1. The selected objective is to minimize an integral of the squared displacements in a region of the design domain for a selected time interval. The optimization results in periodic band-gap structures as demonstrated in Dahl et al (2008) and shown in Fig. 1, with a period depending on the wavelength of the waves propagating through the wave guide. As these results are well known and investigated in details in the literature, study and discussion of the optimized design will be omitted here and the focus will be shifted on the sensitivity analysis.

The gradients of the objective in (9) can be obtained using adjoint analysis as shown in Dahl et al (2008), and are given as

$$\int_0^T\frac{\partial z\left(\boldsymbol{s},\mathbf{u}\right)}{\partial \boldsymbol{s}_e}\mathrm{d}t=\int_0^T\boldsymbol{\lambda}^{\mathsf{T}}\left[\frac{\partial\mathbf{M}\left(\boldsymbol{s}\right)}{\partial\boldsymbol{s}_e}\ddot{\mathbf{u}}+\frac{\partial\mathbf{C}\left(\boldsymbol{s}\right)}{\partial\boldsymbol{s}_e}\dot{\mathbf{u}}+\frac{\partial\mathbf{K}\left(\boldsymbol{s}\right)}{\partial\boldsymbol{s}_e}\mathbf{u}\right]\mathrm{d}t, \tag{15}$$

where the Lagrange multipliers vector $\boldsymbol{\lambda}\left(t\right)=\overline{\boldsymbol{\lambda}}\left(T-\tau\right)$ is obtained as the solution of the following equation

$$\mathbf{M}\ddot{\overline{\boldsymbol{\lambda}}}+\mathbf{C}\dot{\overline{\boldsymbol{\lambda}}}+\mathbf{K}\overline{\boldsymbol{\lambda}}=\frac{\partial z\left(\tau,\mathbf{u}\right)}{\partial\mathbf{u}},\quad\tau\in\left[0,T\right], \tag{16}$$

with initial conditions $\overline{\boldsymbol{\lambda}}=0$ and $\dot{\overline{\boldsymbol{\lambda}}}=0$ and $\tau=T-t$.

2.1 Time integration

As no analytic solution to (10) exists in the general case, the vectors of displacements, velocities and accelerations are obtained numerically at discrete time steps. Here the time derivatives are computed based on finite difference scheme and at the n^{th} time step they are given as

$$\dot{\mathbf{u}}_n=\frac{\mathbf{u}_{n+1}-\mathbf{u}_{n-1}}{2\Delta t}, \tag{17}$$

$$\ddot{\mathbf{u}}_n=\frac{\mathbf{u}_{n+1}-2\mathbf{u}_n+\mathbf{u}_{n-1}}{\Delta t^2}. \tag{18}$$

Inserting (11) and (14) in (10) and rearranging the terms results in

$$\left(\frac{1}{\Delta t^2}\mathbf{M}+\frac{1}{2\Delta t}\mathbf{C}\right)\mathbf{u}_{n+1}=\mathbf{f}_n-\left(\frac{2}{\Delta t^2}\mathbf{M}+\mathbf{K}\right)\mathbf{u}_n-\left(\frac{1}{\Delta t^2}\mathbf{M}-\frac{1}{2\Delta t}\mathbf{C}\right)\mathbf{u}_{n-1}. \tag{19}$$

The above equation provides the solution at time t_{n+1} using the system response at time steps n and $n-1$. The integration starts with $\mathbf{u}_0=0$ and $\mathbf{u}_{-1}=0$. The time step is chosen based on the Courant-Friedrichs-Lewy (CFL) condition

$$\Delta t\leq\Delta t_c=\frac{\Delta x}{c}, \tag{20}$$

where Δx is the distance between the finite element nodes and c is the wave speed. The same scheme is applied for solving the adjoint equation (16). The second derivative for the sensitivity analysis at $t=0$ is computed as $\ddot{\mathbf{u}}=\mathbf{M}^{-1}\mathbf{f}\left(0\right)$.

The Lagrange multipliers sequence can be obtained from (16) by stepping backward in time. First, the forward solution is computed for each discrete point $\mathbf{u}_0,\mathbf{u}_1,\ldots,\mathbf{u}_{N_s-1},\mathbf{u}_{N_s}$, and as second step the adjoint equation is computed with right hand sides depending on the forward solution. As final step the sensitivities are evaluated based on (15). For more details the interested readers are referred to Dahl et al (2008); Elesin et al (2012, 2014); Lazarov et al (2011).

The great advantage of the adjoint approach, compared to for example finite difference derivatives, is that all sensitivities can be evaluated by solving the adjoint problem once. As noted above, however, the same is

true of the reverse mode of automatic differentiation, since the objective function is of type $J : \mathbb{R}^{N_d} \to \mathbb{R}$. For the numerical implementation, the discrete form of the objective J is

$$J = \int_0^T z(\boldsymbol{s}, \boldsymbol{u}) \approx \sum_i z(\boldsymbol{s}, \boldsymbol{u}(t_i))\Delta t. \tag{21}$$

The discretization (19) is sufficiently simple that it can evaluated as a simple stencil type computation, as is illustrated in Algorithm 1.

Algorithm 1 Evaluating the wave propagation objective.

```
u_{-1} = 0, u_0 = 0.
for all t ∈ {0, Δt, 2Δt, ..., T} do
  for all e ∈ {0, ..., N_e − 1} do
    Compute local value u(x_e, t_i) by (19).
  end for
  J ← J + z(s, u)Δt.
end for
```

The key feature of Algorithm 1 is that it is relatively simple to implement without the need of any external linear algebra libraries. This makes it very simple to automatically differentiate the evaluation of the objective function. Using the operator overloading approach, the problem can be simply differentiated in a black box manner without the need of deriving and implementing the adjoint method. As an aside, note that even if a linear algebra library was required, C++ libraries such as Eigen (Guennebaud et al 2010) support linear algebra on arbitrary numeric types, and thus would allow automatic differentiation as well.

In order to allow the application of operator overloading AD, our implementation of Algorithm 1 was converted into a C++ template, thus allowing the implementation to use the numeric types provided by CoDiPack. After this, the code must perform some calls to the tape type provided by CoDiPack, before and after the call to the function evaluating Algorithm 1. These additional calls add very little code and are described in the CoDiPack tutorial (CoDiPack website 2016). The differentiation procedure using CoDiPack yields identical sensitivities to those yielded by evaluating the adjoint expression (15). Comparative performance measures are shown in Table 1. The code was compiled with GCC 5.4.0 with `-O2`, and the performance was measured on an Intel Core i7-3720QM processor.

As expected the memory and the computational time grow proportional to the number of the time steps for the AD and the hand coded example. Comparing the performance of AD with CoDiPack to the hand coded adjoint, CoDiPack is roughly 1.5 to 1.8 times slower. Considering that the development time needed to obtain the derivative code is essentially zero, this seems like a modest price to pay, though this would of course depend on the specifics of the problem and the performance requirements. The memory requirements of the AD solution, however, is more than an order of magnitude greater than the hand coded equivalent. This is because the tape structure implemented by CoDiPack must store, in addition to all solution states $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N_s}$, all operations needed by the computation in order to reverse them. Whether this memory requirement is an intractable issue depends on the nature and size of the problem one wishes to solve. For a research problem such as this, the memory requirement of AD is available on many modern personal computers, and the advantage of being able to differentiate a code without spending any significant time deriving, implementing, and debugging an adjoint solver can hardly be emphasized enough. Even in cases where AD does not scale to the desired problem size, it would still be a useful tool for prototyping the optimization problem and verifying the derivation and implementation of an adjoint code. As a final point, note that the large memory footprint of CoDiPack is due to the fact that we are solving a transient problem. This means that the tape structure redundantly stores the time stepping operations once for each time step taken. Thus, steady state type problems would require significantly less memory. In addition, CoDiPack is actively developed and future optimizations might address this issue.

In the above discussion we considered a problem which readily lent itself to black box automatic differentiation. In the next section, we will consider a more complicated example, in which automatic differentiation is applied to an already existing parallel code.

3 Application to lattice Boltzmann

In this section, we will focus on a more involved example of automatic differentiation applied to the lattice Boltzmann method (LBM). While there has been work to apply AD to an already existing parallel LBM code (Krause and Heuveline 2013), the LBM implementation presented here uses a few components from the open source topology optimization code presented by Aage et al (2015), and thus uses the PETSc library (Balay et al 2016a,b) for execution in parallel. In this case, neither source transformation (i.e. with Tapenade) nor operator overloading (i.e. with CoDiPack) are naively applicable, since they do not interface directly with PETSc. A different strategy than simple black box differentiation is required (Sagebaum et al 2013). For

2500 timesteps		
	Automatic differentiation	Hand coded adjoint
Memory	1.0 Gb	–
Wall time	6 s	4.2 s
Relative time	1.43	1
5000 timesteps		
	Automatic differentiation	Hand coded adjoint
Memory	2.1 Gb	0.07 Gb
Wall time	13 s	8.5 s
Relative time	1.52	1
10000 timesteps		
	Automatic differentiation	Hand coded adjoint
Memory	4.2 Gb	0.15 Gb
Wall time	26 s	16 s
Relative time	1.625	1
20000 timesteps		
	Automatic differentiation	Hand coded adjoint
Memory	8.5 Gb	0.27 Gb
Wall time	50.53 s	32.46 s
Relative time	1.55	1
30000 timesteps		
	Automatic differentiation	Hand coded adjoint
Memory	12.8 Gb	0.43 Gb
Wall time	88 s	48 s
Relative time	1.83	1

Table 1 Performance measurements comparing AD to the hand coded adjoint for the wave propagation problem. The number of elements is $N_e = 900$. The memory utilized for AD can be significantly reduced to the level of the hand coded adjoint by introducing checkpointing as discussed in section 3.2.

lattice Boltzmann, it is possible to derive an adjoint method in which the local operations can be differentiated with AD. In this way, the AD code is only invoked within the main loop, which decouples the code from external library calls.

3.1 The lattice Boltzmann equation

The lattice Boltzmann method is a method for computing fluid flows based on kinetic theory, rather than continuum dynamics. A thorough introduction is beyond the scope of this paper, but the interested reader is referred to e.g. the book by Succi (2001). LBM is an explicit time-stepping method, based on the equation

$$\begin{aligned} &f_\alpha(\boldsymbol{x}_i + \boldsymbol{e}_\alpha \Delta x, t + \Delta t) = \Omega[\boldsymbol{f}(\boldsymbol{x}_i, t)], \\ &\alpha \in \{0, \dots, N_v - 1\}, \end{aligned} \tag{22}$$

where $\boldsymbol{f} \in \mathbb{R}^{N_v}$ is a set of distribution values associated with a discrete set of particle velocities $\boldsymbol{e}_\alpha$. The right hand side models particle collisions and is known as the *collision operator*. There are numerous different collision operators available in the literature (Bhatnagar et al 1954; D'Humieres 1994; Geier et al 2006; Latt and Chopard 2006), and a large class of lattice Boltzmann models differ only in the collision operator, while the left-hand side—known as the *streaming step*—remains unchanged. The collision operator is in general highly non-linear in $\boldsymbol{f}$; indeed, this is part of the reason automatic differentiation is attractive for the lattice Boltzmann method. For the purpose of density based topology optimization, (22) is modified as follows:

$$\begin{aligned} &f_\alpha(\boldsymbol{x}_i + \boldsymbol{e}_\alpha \Delta x, t + \Delta t) = \tilde{\Omega}[\boldsymbol{f}(\boldsymbol{x}_i, t), s(\boldsymbol{x}_i)], \\ &\alpha \in \{0, \dots, N_v - 1\}, \end{aligned} \tag{23}$$

where $s(\boldsymbol{x}_i) \equiv s_i$ determines whether the grid point $\boldsymbol{x}_i$ is a fluid or solid node. This modification of the collision step is to enforce an immersed no-slip boundary in the solid part of the domain. Again, numerous models to achieve this are available in the literature (Ladd and Verberg 2001; Spaid and Phelan 1997; Zhu and Ma 2013), and the modification is typically orthogonal to the choice of "base" operator, leaving a high number of possibly combinations that are all valid collision operators.

The macroscopic variables of the flow governed by equation (23) can be computed by

$$\rho(\boldsymbol{x}_i, t) = \sum_\alpha f_\alpha(\boldsymbol{x}_i, t), \tag{24}$$

$$\rho(\boldsymbol{x}_i, t)\boldsymbol{u}(\boldsymbol{x}_i, t) = \sum_\alpha \boldsymbol{e}_\alpha f_\alpha(\boldsymbol{x}_i, t), \tag{25}$$

$$p(\boldsymbol{x}_i, t) = c_s^2 \rho(\boldsymbol{x}_i, t), \tag{26}$$

where $\rho, p, \boldsymbol{u}$ are the macroscopic density, pressure, and velocity, respectively. The lattice Boltzmann method is

a weakly compressible method, and the macroscopic pressure is proportional to the macroscopic density, with a proportionality constant equal to c_s^2, the speed of sound squared. The numerical value of this constant depends on the choice of velocity discretization.

One attractive feature of the lattice Boltzmann method is that the algorithm has high spatial locality: the collision step requires only local information while the streaming step requires only nearest neighbor information. This makes it ideally suited for execution in parallel.

Time stepping in the LBM can be executed in either a *collide and stream* fashion, in which the collision step is executed followed by the streaming step, or conversely in a *stream and collide* fashion. For the purpose of topology optimization, we choose the stream and collide approach. The reason for this is that the objective is a function of the macroscopic values, which are evaluated during the collision step. Hence, by performing stream and collide from timestep n to $n+1$, the macroscopic variables are also in the correct state at step $n+1$. A function of the macroscopic variables can then be conveniently evaluated following the stream and collide procedure.

In residual form the LB scheme can be written:

$$R_\alpha^{\text{stream}}(\boldsymbol{x}_i,t) = f_\alpha^{\text{collision}}(\boldsymbol{x}_i+\boldsymbol{e}_\alpha,t+\Delta t) - f_\alpha^{\text{stream}}(\boldsymbol{x}_i,t) = 0, \tag{27a}$$

$$\boldsymbol{R}^{\text{bc}}(\boldsymbol{x}_i,t) = \psi[\boldsymbol{f}^{\text{stream}}(\boldsymbol{x}_i,t)] - \boldsymbol{f}(\boldsymbol{x}_i,t) = 0, \tag{27b}$$

$$\boldsymbol{R}^{\text{collision}}(\boldsymbol{x}_i,s_i,t) = \tilde{\Omega}[\boldsymbol{f}(\boldsymbol{x}_i,t),s_i] - \boldsymbol{f}^{\text{collision}}(\boldsymbol{x}_i,t) = 0. \tag{27c}$$

Above, $\boldsymbol{f}$ denotes the initial state of distributions at timestep t, $\boldsymbol{f}^{\text{collision}}$ denotes the post-collision state, and $\boldsymbol{f}^{\text{stream}}$ denotes the post-streaming state. On interior nodes $\boldsymbol{f}=\boldsymbol{f}^{\text{stream}}$; on boundary nodes, however, there are unknown distribution values, which are computed in the boundary value step (27a). Here ψ simply denotes a generic boundary condition operator. For the LBM, there are many different operators available for different kinds of boundaries (Inamuro et al 1995; Junk and Yang 2008; Latt et al 2008; Zou and He 1997).

3.2 Automatic differentiation of lattice Boltzmann

As mentioned above, because the LB code relies on an external library, it is not feasible to differentiate the code in a black box manner. Instead, the discrete adjoint method is applied to obtain an adjoint lattice Boltzmann method in which the local collision step can be evaluated with automatic differentiation. A similar derivation was given by Łaniewski Wołłk and Rokicki (2016).

Following Kreissl et al (2011), we consider an objective function for unsteady flow of the following form

$$J = \sum_{t=0}^{N_t} z(t,\boldsymbol{f}_t,\boldsymbol{s}), \tag{28}$$

where N_t is the number of time steps, $\boldsymbol{f}_t = [\boldsymbol{f}(\boldsymbol{x}_0,t),\boldsymbol{f}(\boldsymbol{x}_1,t),\ldots]$ is the vector of state variables (i.e. the LBM distributions) at timestep t, and $\boldsymbol{s}=[s_0,s_1,\ldots]$ is the vector of design variables. To derive the adjoint LBM, Lagrange multipliers are added to (28):

$$\hat{J} = \sum_{t=0}^{N_t} z(t,\boldsymbol{f}_t,\boldsymbol{s}) + \boldsymbol{\lambda}_t^T\boldsymbol{R}_t^{\text{stream}} + \boldsymbol{\sigma}_t^T\boldsymbol{R}_t^{\text{bc}} + \boldsymbol{\tau}_t^T\boldsymbol{R}_t^{\text{collision}}. \tag{29}$$

Taking the derivative with respect to the design variable s_i yields:

$$\frac{\mathrm{d}\hat{J}}{\mathrm{d}s_i} = \frac{\partial\hat{J}}{\partial s_i} + \sum_{t=0}^{N_t}\frac{\partial\hat{J}}{\partial\boldsymbol{f}_t}\frac{\partial\boldsymbol{f}_t}{\partial s_i} + \frac{\partial\hat{J}}{\partial\boldsymbol{f}_t^{\text{collision}}}\frac{\partial\boldsymbol{f}_t^{\text{collision}}}{\partial s_i} + \frac{\partial\hat{J}}{\partial\boldsymbol{f}_t^{\text{stream}}}\frac{\partial\boldsymbol{f}_t^{\text{stream}}}{\partial s_i}. \tag{30}$$

For an optimal design, we must have $\mathrm{d}\hat{J}/\mathrm{d}s_i = 0, \forall i$. Since each term in (30) is mutually independent, this implies that each summand must be zero.

From the residuals (27), we then have:

$$\sum_{t=0}^{N_t}\frac{\partial\hat{J}}{\partial\boldsymbol{f}_t}\frac{\partial\boldsymbol{f}_t}{\partial s_i} = \sum_{t=0}^{N_t}\left(\boldsymbol{\tau}_t^T\frac{\partial\tilde{\boldsymbol{\Omega}}}{\partial\boldsymbol{f}_t} - \boldsymbol{\sigma}_t^T\boldsymbol{I} + \frac{\partial z}{\partial\boldsymbol{f}_t}\right)\frac{\partial\boldsymbol{f}_t}{\partial s_i} = 0, \tag{31}$$

where $\boldsymbol{I}$ is the identity matrix. Since the collision $\boldsymbol{\Omega}$ is purely local, this implies

$$\boldsymbol{\sigma}(\boldsymbol{x}_i,t)^T = \boldsymbol{\tau}(\boldsymbol{x}_i,t)^T\frac{\partial\tilde{\Omega}[\boldsymbol{f}(\boldsymbol{x}_i,t),s_i]}{\partial\boldsymbol{f}(\boldsymbol{x}_i,t)} + \frac{\partial z(t,\boldsymbol{f}_t,\boldsymbol{s})}{\partial\boldsymbol{f}(\boldsymbol{x}_i,t)}. \tag{32}$$

This is the adjoint collision step. Notice that the first summand on the right-hand side of (32) is of the form (8), meaning that it can be evaluated exactly with one computation of the AD reverse mode.

Continuing, we further have:

$$\sum_{t=0}^{N_t} \frac{\partial \hat{J}}{\partial \boldsymbol{f}_t^{\text{stream}}} \frac{\partial \boldsymbol{f}_t^{\text{stream}}}{\partial s_i} = \sum_{t=0}^{N_t} \left(\boldsymbol{\sigma}_t^T \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{f}_t^{\text{stream}}} - \boldsymbol{\lambda}_t^T \right) \frac{\partial \boldsymbol{f}_t^{\text{stream}}}{\partial s_i}, \tag{33}$$

$$\boldsymbol{\lambda}(\boldsymbol{x}_i, t)^T = \boldsymbol{\sigma}(\boldsymbol{x}_i, t)^T \frac{\partial \psi[\boldsymbol{f}^{\text{stream}}(\boldsymbol{x}_i, t)]}{\partial \boldsymbol{f}^{\text{stream}}(\boldsymbol{x}_i, t)}, \tag{34}$$

which is the adjoint boundary step, assuming the boundary function ψ is purely local. This could again be evaluated by AD, which would be advantageous for complicated boundary conditions such as the regularized boundary conditions (Latt et al 2008). For simpler boundary conditions such as those presented by Zou and He (1997), or the frequently applied "bounce back" no-slip condition, it is quite simple to derive this step by hand.

Differentiation of the final step leads to the adjoint streaming step, which was shown by Liu et al (2014) to be given by

$$\tau_\alpha(\boldsymbol{x}_i, t) = \lambda_\alpha(\boldsymbol{x}_i - \boldsymbol{e}_\alpha, t - \Delta t), \tag{35}$$

that is, the adjoint streaming is backwards in time and in the opposite direction of the primal streaming. Finally, the sensitivities can be evaluated by

$$\frac{\partial \hat{J}}{\partial s_i} = \sum_{t=0}^{N_t} \frac{\partial z}{\partial s_i} + \boldsymbol{\tau}_t^T \frac{\partial \tilde{\boldsymbol{\Omega}}}{\partial s_i} \tag{36}$$

$$= \sum_{t=0}^{N_t} \frac{\partial z}{\partial s_i} + \tau(\boldsymbol{x}_i, t)^T \frac{\partial \tilde{\Omega}[\boldsymbol{f}(\boldsymbol{x}_i, t), s_i]}{\partial s_i}, \tag{37}$$

with the final equality again being due to the local nature of the collision operator Ω. This completes the adjoint lattice Boltzmann method, its implementation is summarized by pseudo code in Algorithm 2.

The adjoint lattice Boltzmann algorithm step backwards through time to evaluate the Lagrange multipliers and thus the sensitivities. Note that at each timestep, the primal vector $\boldsymbol{f}_t$ must be known in order to evaluate the adjoint lattice Boltzmann step. As a consequence, the full time history of the primal solver must be available. Naively, this means that the full history must be stored in memory. While such a strategy is feasible for small problems, it does not scale well. As an alternative, parts of the history can be recomputed during the adjoint evaluation. With this strategy, only selected time steps are stored in memory. These time steps are typically referred to as *checkpoints*. The rest of the time steps are then recomputed starting from the nearest checkpoint as they are needed. The papers by Griewank and Walther (2000) and Wang et al (2009) both describe provably optimal algorithms for checkpoint placement. With these algorithms, the cost of re-computation grows only logarithmically with the memory saved. For example, allocating 20 checkpoints for an objective requiring 200 time steps to evaluate reduces the memory requirement by an order of magnitude compared to the naive approach, but only increases the computational cost of the adjoint evaluation by a factor of $\log 10$.

Algorithm 2 Adjoint lattice Boltzmann with AD.

```
for all t ∈ {N_t, ..., 0} do
    Obtain f_t by reading from memory or performing necessary re-computation.
    for all x_i, i ∈ {0, ..., N_x − 1} do
        Compute adjoint collision step by equation (32).
        Add contribution to sensitivity dJ/ds_i by equation (37).
        if x_i is a boundary node then
            Compute adjoint boundary conditions by (34).
        end if
        for all x_i, i ∈ {0, ..., N_x − 1} do
            Perform adjoint streaming by (35).
        end for
    end for
end for
```

Note that Algorithm 2 is executed in collide and stream order. This is a consequence of our choice of the stream and collide order for the primal solver. Had we chosen collide and stream for the primal solver, the adjoint algorithm would have to be executed in stream and collide order.

It should be emphasized that Algorithm 2 can be used to differentiate a large class of LB models, as long as the model follows the basic structure of a local collision step and a shifting streaming step. More complicated models which follow this basic structure include thermal lattice Boltzmann (Bartoloni et al 1993; Guo et al 2002; Mezrhab et al 2010), as well as lattice Boltzmann for multi-component flow (Asinari 2006; Parker 2008).

3.3 An example problem

The main challenge in implementing the adjoint LBM introduced above is the evaluation of the adjoint collision step (32). Of course, the collision operator could be differentiated by hand, but as noted above, the equation can be evaluated by applying the reverse mode of automatic differentiation. In this section, we will test our implementation against an example problem, followed by an evaluation of the performance of different AD implementations.

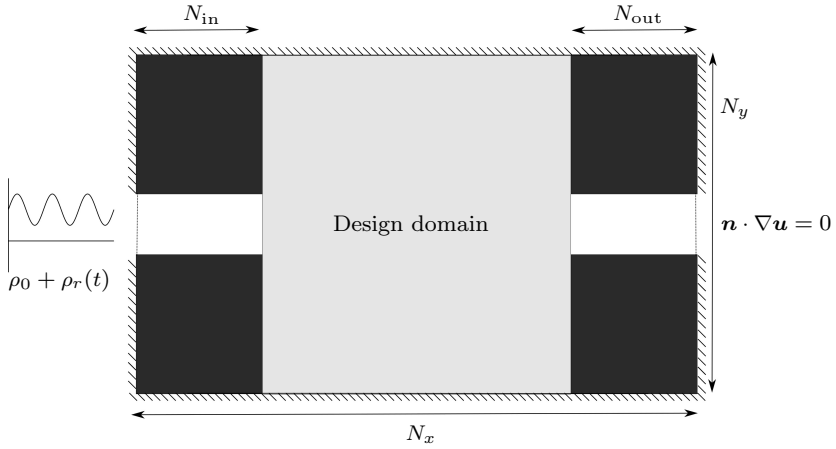


Fig. 3 Computational domain for the pressure diode problem.

For the sample problem, the collision operator Ω applied is the multiple relaxation time (MRT) operator (D'Humieres 1994), operating on the common D2Q9 lattice (nine discrete velocities in two dimensions). For this lattice, the velocities are given by

$$[\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_4, \boldsymbol{e}_5, \boldsymbol{e}_6, \boldsymbol{e}_7, \boldsymbol{e}_8] = \frac{\Delta x}{\Delta t} \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}, \tag{38}$$

the set of velocities defined by the D2Q9 lattice is illustrated in Fig. 2.

In order to enforce the no-slip condition on the solid part of the domain, we use the partial bounce back collision operator introduced by Zhu and Ma (2013). In this model, the base collision operator Ω is modified to

$$\begin{aligned} \tilde{\Omega}[\boldsymbol{f}(\boldsymbol{x}_i, t), s_i]_\alpha =& \Omega[\boldsymbol{f}(\boldsymbol{x}_i, t)] + \frac{1}{2} g(s_i) \times \\ & (\Omega[\boldsymbol{f}(\boldsymbol{x}_i, t)]_{-\alpha} - \Omega[\boldsymbol{f}(\boldsymbol{x}_i, t)]_\alpha), \end{aligned} \tag{39}$$

where the index $-\alpha$ indicates the discrete velocity opposite to the index α, i.e. $\boldsymbol{e}_{-1} = \boldsymbol{e}_3$; the function $g(s_i)$ is continuous and satisfies $g(0) = 1$ and $g(1) = 0$, so that $s_i = 0$ corresponds to a solid node, while $s_i = 1$ corresponds to a fluid node. Here, we use the following convex function introduced by Borrvall and Petersson (2003):

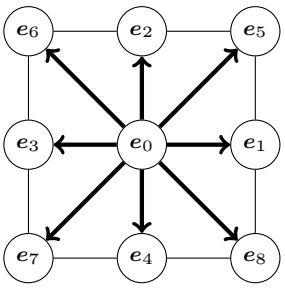


Fig. 2 The D2Q9 model.

$$g(s_i) = 1 - s_i \frac{1 + \gamma}{s_i + \gamma}, \tag{40}$$

where γ is an adjustable parameter which allows penalization of intermediate values of s_i. Increasing γ increases the penalization of intermediate values.

The example problem considered is an unsteady flow problem with an objective function of the form (28). The computational domain for the problem is shown in Fig. 3. The problem is inspired by the work on fluid diodes by Lin et al (2015).

The computational domain consists of two narrow channels, the left side with prescribed density (and therefore pressure, since $\rho \propto p$ in LBM), and the right side with a Neumann boundary on the velocity. The enforced density on the left is oscillating, with oscillations given by

$$\rho_{\text{oscillating}}(t) = \rho_0 + \rho_r(t) = \rho_0 + \Delta\rho \sin\left(\frac{2\pi t}{\omega}\right), \tag{41}$$

here, $\Delta\rho$ is the amplitude of the oscillation, and ω is the period. We now seek to maximize the average outflow at the right end, subject to a volume constraint on the amount of fluid in the design domain. That is, the optimization problem is formulated as:

$$\begin{aligned} &\min_{\boldsymbol{s}} J = -\frac{1}{N_t} \sum_{t=0}^{N_t} \bar{u}_x, \\ &\text{s.t.} \begin{cases} \dfrac{1}{N_{\boldsymbol{s}}} \displaystyle\sum_i s_i - V_{\text{fluid}} \leq 0, \\ \boldsymbol{f}_t \text{ satisfies (27)}, \end{cases} \end{aligned} \tag{42}$$

where V_{fluid} is the allowed fraction of fluid in the design domain, and $\bar{u}_x$ is the spatially averaged x-component of the velocity at the right outlet. In order to regularize

N_x	350	N_y	125
N_{in}	75	N_t	20000
ρ_0	1	$\Delta\rho$	0.01
ω	1000	V_{fluid}	0.6
γ	1	Filter radius	6

Table 2 Numerical parameters for the example problem.

the design, and obtain a fully black and white solution, the projection filter (Guest et al 2004) is applied. To compute the Reynolds number, the characteristic length is defined as $L = N_{\text{in}}$, and the characteristic velocity is taken to be $\boldsymbol{u}_{\text{characteristic}} = 0.01$. The choice of characteristic velocity is somewhat arbitrary, since no set velocity is directly imposed anywhere in the domain, but agrees well with the observed order of magnitude of velocities in the final designs. The remaining numerical parameters used are listed in Table 2.

Two example designs at different Reynolds numbers are shown in Fig. 4. Both have the same basic structure, but higher Reynolds number results in slightly more intricate side channels in the final design. In order to better understand the working principle of the designs, sample streamlines are shown in Fig. 5, both for the case of the oscillatory term in equation (41) being negative ($\rho_r > 0$), and positive ($\rho_r < 0$). From the figure, it is observed that even though the oscillating pressure on the left side results in fluid periodically flowing both in and out at the boundary, the right boundary only ever acts as an outflow. It appears that the side "arms" of the design act as a deposit for fluid during the outflow phase of the left boundary; this deposited fluid then flows towards the desired outlet when the pressure oscillations reverse. In Fig. 6, the average outflow is plotted as a function of time. It is observed that the cyclic behaviour observed in Fig. 5 does indeed repeat throughout the whole time history.

3.4 Performance of AD implementation

To close this section, the performance of different AD implementations will be reported. The performance is measured according to the following methodology: since reverse AD is applied only in the adjoint collision step (32), we will only measure the computational time of this step. The adjoint collision step is implemented in a simple C++ for loop, no attempts have been made at optimization for memory accesses. The performance metric will be the average collisions per second (CPS) in a single iteration of the example problem presented above. Since the adjoint collision step is purely local, we will consider only the single core performance and thus ignore any parallel message passing overhead. The performance is measured on an Intel Xeon X5660 processor.

In addition to the MRT collision operator used above, we will consider the commonly used Bhatnagar-Gross-Krook (BGK) collision operator Bhatnagar et al (1954), as well as the more recent cascaded collision operator by Geier et al (2006). Both operator overloading and source transformation implementations of the adjoint collision (32) will be considered. For operator overloading, CoDiPack will be used. For source transformation, the online tool Tapenade will be used. For Tapenade, two versions will be considered: the "raw" source transformation output, and a version of the source transformed output which has been hand optimized. All kernels have been compiled with GCC 4.8.5 with `-O3`. The results of the performance measurements are listed in Table 3.

As is apparent from Table 3, unsurprisingly, the best performance also comes from the implementation which requires the most effort. While the CoDiPack implementation cannot compete with Tapenade in terms of speed, it should once again be reiterated that using Tapenade involves a trade-off between implementation time and running time. Even if a good optimized collision routine is implemented with the help of Tapenade, any changes in the source code for the primal collision step will not be reflected in the adjoint code. Conversely, with CoDiPack, any optimizations made to the primal collision source code will immediately result in better adjoint performance with no additional implementation effort.

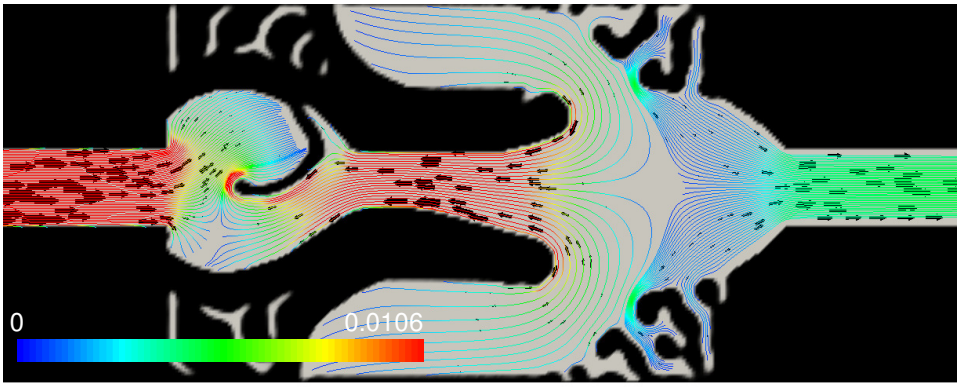
4 Discussion and conclusion

In this paper, we have demonstrated the application of automatic differentiation to two different classes of problems for topology optimization. While the AD promise of completely black box differentiation of numerical codes is certainly tantalizing, achieving this does require that the code has been written with the application of AD in mind. For codes where this is not the case, some additional implementation work will be necessary. At best, it is simply a matter of parametrizing core routines to accept generic numeric types (e.g. turning core routines into templates). For more complicated codes, which might have external dependencies which are unrealistic or even impossible to modify, a significantly greater implementation effort could be required. Whether this time investment is worth it will of course be project dependent.

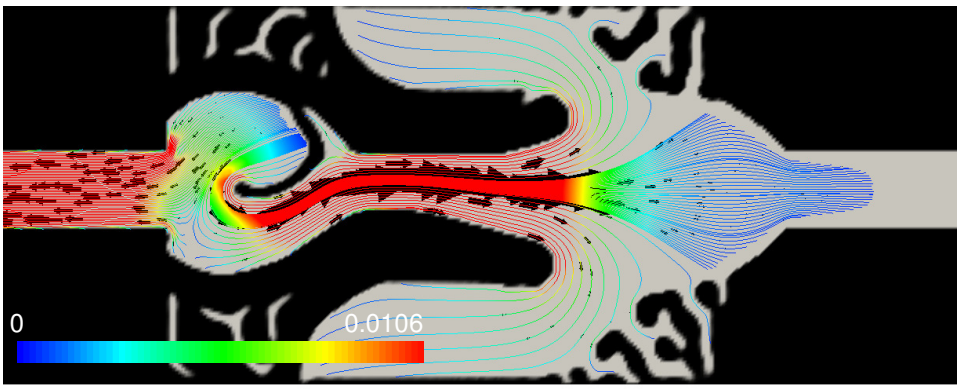
While the above considerations does limit the applicability of AD to some extent, many research codes are developed from scratch in order to solve a single

(a) Re = 100. (b) Re = 250.

Fig. 4 Example results at different Reynolds numbers.



(a) $\rho_r > 0$.



(b) $\rho_r < 0$.

Fig. 5 Sample streamlines during the inflow and outflow phase for the result at Re = 250.

Problem size: 350×125, 20000 timesteps			
	BGK	MRT	Cascaded
CoDiPack	1.12×10^6 CPS	0.631×10^6 CPS	0.481×10^6 CPS
Tapenade	4.18×10^6 CPS	4.32×10^6 CPS	1.17×10^6 CPS
Tapenade (optimized)	12.27×10^6 CPS	7.56×10^6 CPS	4.23×10^6 CPS

Table 3 Results of performance measurements for adjoint LBM with AD. Higher CPS (collisions per second) is better.

well-defined problem. In these cases, getting the derivatives of a function for "free" can greatly decrease the time required to solve a particular problem; even in cases where black box differentiation is not possible, AD might be still be applicable with a bit more up front work. This was demonstrated in the lattice Boltzmann example above. Here, some work was required to derive and implement the AD supported adjoint method,

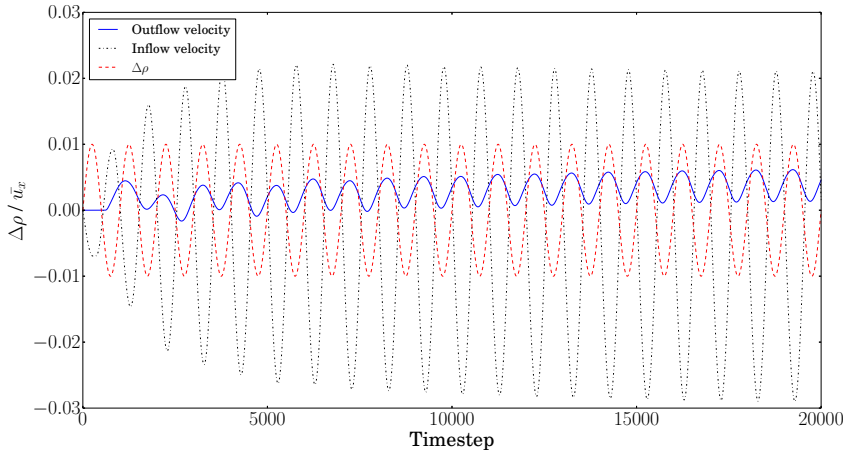


Fig. 6 Average outflow velocity of the optimized design at $\mathrm{Re} = 250$ as a function of time. Also shown are the average inflow velocity, and the density variation $\Delta\rho$.

but once this was done, it became possible to differentiate any lattice Boltzmann type method with little additional work.

The final point to consider is the issue of performance. In both problems presented, there is a trade-off between performance and development time; in both cases it is possible to improve performance by implementing a hand tuned adjoint code (either by derivation or by optimization of the output from Tapenade). However, even if these performance improvements were strictly necessary in order to solve the problem within a realistic time, the less performant version would still be useful for prototyping and validation. During development of the optimized Tapenade routines for lattice Boltzmann, the CoDiPack adjoint collision implementation was used as a reference known to give the correct answer. This greatly eased development, since any mistakes introduced during the tuning of the code were immediately caught.

As with all things in software development, automatic differentiation is a technique which comes with advantages and disadvantages. In the view of the authors, it is a powerful tool that can be used to great advantage in many types of problems in topology optimization, and should be considered as a useful supplement to hand derived adjoints.

Acknowledgements The first author would like to acknowledge the generous help and fruitful discussion offered by Emre Özkaya and Tim Albring at TU Kaiserslautern. The first and the last authors acknowledge the financial support received from the TopTen project sponsored by the Danish Council for Independent Research (DFF-4005-00320).

References

Aage N, Andreassen E, Lazarov BS (2015) Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. Structural and Multidisciplinary Optimization 51(3):565–572, DOI 10.1007/s00158-014-1157-0

Albring T, Sagebaum M, Gauger N (2015a) Development of a consistent discrete adjoint solver in an evolving aerodynamic design framework. AIAA 2015-3240

Albring T, Zhou B, Gauger N, Sagebaum M (2015b) An aerodynamic design framework based on algorithmic differentiation. ERCOFTAC Bulletin 102:10–16

Albring T, Sagebaum M, Gauger NR (2016) Efficient aerodynamic design using the discrete adjoint method in su2. 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference

Asinari P (2006) Semi-implicit-linearized multiple-relaxation-time formulation of lattice Boltzmann schemes for mixture modeling. Physical Review E 73(5):056,705, DOI 10.1103/PhysRevE.73.056705

AutoDiff website (2016) autodiff.org: Community portal for automatic differentiation. URL `http://www.autodiff.org`, accessed: 2016-10-18

Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, Dalcin L, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Rupp K, Smith BF, Zampini S, Zhang H, Zhang H (2016a) PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.7, Argonne National Laboratory, URL `http://www.mcs.anl.gov/petsc`

Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, Dalcin L, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Rupp K, Smith BF, Zampini S, Zhang H, Zhang H (2016b) PETSc Web page. URL `http://www.mcs.anl.gov/petsc`

Bartoloni A, Battista C, Cabasino S, Paolucci P, Pech J, Sarno S, Todesco G, Torelli M, Tross W, Vicini P, Benzi R, Cabibbo N, Massaioli F, Tripiccione R (1993) LBE simulations of Rayleigh-Benard convection on the APE100 parallel processor. International Journal of Modern Physics C-physics and Computers 4(5):993–1006, DOI 10.1142/S012918319300077X

Bendsøe MP, Sigmund O (2004) Topology optimization: theory, methods and applications. Springer

Bhatnagar PL, Gross EP, Krook M (1954) A model for collision processes in gases. I: Small amplitude processes in charged and neutral one-component systems. Physical Review 94(3)

Borrvall T, Petersson J (2003) Topology optimization of fluids in stokes flow. International Journal for Numerical Methods in Fluids 41(1):77–107, DOI 10.1002/fld.426

CoDiPack website (2016) CoDiPack—code differentiation package. URL http://www.scicomp.uni-kl.de/software/codi/, accessed: 2016-10-18

Dahl J, Jensen JS, Sigmund O (2008) Topology optimization for transient wave propagation problems in one dimension. Structural and Multidisciplinary Optimization 36:585–595

D'Humieres D (1994) Generalized lattice Boltzmann equations. Progress in Astronautics and Aeronautics 159:450–458

Elesin Y, Lazarov B, Jensen J, Sigmund O (2012) Design of robust and efficient photonic switches using topology optimization. Photonics and Nanostructures - Fundamentals and Applications 10(1):153 – 165, DOI 10.1016/j.photonics.2011.10.003, URL http://www.sciencedirect.com/science/article/pii/S1569441011001106

Elesin Y, Lazarov B, Jensen J, Sigmund O (2014) Time domain topology optimization of 3d nanophotonic devices. Photonics and Nanostructures - Fundamentals and Applications 12(1):23 – 33, DOI http://dx.doi.org/10.1016/j.photonics.2013.07.008, URL http://www.sciencedirect.com/science/article/pii/S1569441013000497

Geier M, Greiner A, Korvink JG (2006) Cascaded digital lattice Boltzmann automata for high Reynolds number flow. Physical Review E 73(6):066,705, DOI 10.1103/PhysRevE.73.066705

Griewank A, Walther A (2000) Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. Acm Transactions on Mathematical Software 26(1):19–45, 19–45, DOI 10.1145/347837.347846

Griewank A, Walther A (2008) Automatic differentiation of algorithms. SIAM,

Guennebaud G, Jacob B, et al (2010) Eigen v3. http://eigen.tuxfamily.org

Guest JK, Prévost JH, Belytschko T (2004) Achieving minimum length scale in topology optimization using nodal design variables and projection functions. International Journal for Numerical Methods in Engineering 61(2):238–254, DOI 10.1002/nme.1064, URL http://dx.doi.org/10.1002/nme.1064

Guo Z, Shi B, Zheng C (2002) A coupled lattice BGK model for the Boussinesq equations. International Journal for Numerical Methods in Fluids 39(4):325–342, DOI 10.1002/fld.337

Hascoët L, Pascual V (2013) The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. ACM Transactions On Mathematical Software 39(3), URL http://dx.doi.org/10.1145/2450153.2450158

Hogan RJ (2014) Fast reverse-mode automatic differentiation using expression templates in C++. ACM Transactions on Mathematical Software (toms) 40(4):1–16, DOI 10.1145/2560359

Inamuro T, Yoshino M, Ogino F (1995) A non-slip boundary-condition for lattice boltzmann simulations. Physics of Fluids 7(12):2928–2930, DOI 10.1063/1.868766

Junk M, Yang Z (2008) Outflow boundary conditions for the lattice Boltzmann method. Progress in Computational Fluid Dynamics 8(1-4):38–48, DOI 10.1504/PCFD.2008.018077

Krause MJ, Heuveline V (2013) Parallel fluid flow control and optimisation with lattice Boltzmann methods and automatic differentiation. Computers and Fluids 80:28–36, DOI http://dx.doi.org/10.1016/j.compfluid.2012.07.026, URL http://www.sciencedirect.com/science/article/pii/S0045793012002940, selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011

Kreissl S, Pingen G, Maute K (2011) Topology optimization for unsteady flow. International Journal for Numerical Methods in Engineering 87(13):1229–1253, DOI 10.1002/nme.3151

Ladd A, Verberg R (2001) Lattice-Boltzmann simulations of particle-fluid suspensions. Journal of Statistical Physics 104(5-6):1191–1251, DOI 10.1023/A:1010414013942

Latt J, Chopard B (2006) Lattice Boltzmann method with regularized pre-collision distribution functions. Mathematics and Computers in Simulation 72(2-6):165–168, DOI 10.1016/j.matcom.2006.05.017

Latt J, Chopard B, Malaspinas O, Deville M, Michler A (2008) Straight velocity boundaries in the lattice Boltzmann method. Physical Review E 77(5):056,703, DOI 10.1103/PhysRevE.77.056703

Lazarov B, Matzen R, Elesin Y (2011) Topology optimization of pulse shaping filters using the hilbert transform envelope extraction. Structural and Multidisciplinary Optimization 44:409–419, 10.1007/s00158-011-0642-y

Lin S, Zhao L, Guest JK, Weihs TP, Liu Z (2015) Topology optimization of fixed-geometry fluid diodes. Journal of Mechanical Design 137(8):081,402, DOI 10.1115/1.4030297

Liu G, Geier M, Liu Z, Krafczyk M, Chen T (2014) Discrete adjoint sensitivity analysis for fluid flow topology optimization based on the generalized lattice Boltzmann method. Computers and Mathematics With Applications 68(10):1374–1392, DOI 10.1016/j.camwa.2014.09.002

Mezrhab A, Moussaoui MA, Jami M, Naji H, Bouzidi M (2010) Double MRT thermal lattice Boltzmann method for simulating convective flows. Physics Letters a 374(34):3499–3507, DOI 10.1016/j.physleta.2010.06.059

Nemili A, Özkaya E, Gauger NR, Kramer F, Höll T, Thiele F (2014) Optimal design of active flow control for a complex high-lift configuration. In: Proceedings of 7th AIAA Flow Control Conference, 2014-2515

Nørgaard S, Sigmund O, Lazarov B (2016) Topology optimization of unsteady flow problems using the lattice Boltzmann method. Journal of Computational Physics 307:291 – 307, DOI 10.1016/j.jcp.2015.12.023

Parker J (2008) A novel lattice Boltzmann method for treatment of multicomponent convection, diffusion, and reaction phenomena in multiphase systems. PhD thesis, Oregon State University

Sagebaum M, Gauger NR, Naumann U, Lotz J, Leppkes K (2013) Algorithmic differentiation of a complex C++ code with underlying libraries. Procedia Computer Science 18:208–217, DOI 10.1016/j.procs.2013.05.184

Sigmund O, Maute K (2013) Topology optimization approaches. Structural and Multidisciplinary Optimization 48(6):1031–1055, DOI 10.1007/s00158-013-0978-6, URL http://dx.doi.org/10.1007/s00158-013-0978-6

Spaid M, Phelan F (1997) Lattice Boltzmann methods for modeling microscale flow in fibrous porous media. Physics of Fluids 9(9):2468–2474, DOI 10.1063/1.869392

Succi S (2001) The lattice Boltzmann equation for fluid dynamics and beyond. Oxford University Press

Tapenade website (2016) Tapenade on-line automatic differentiation engine. URL `http://www-tapenade.inria.fr:8080/tapenade/index.jsp`, accessed: 2016-10-18

Wang Q, Moin P, Iaccarino G (2009) Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. Siam Journal on Scientific Computing 31(4):2549–2567, DOI 10.1137/080727890

Łaniewski Wołłk L, Rokicki J (2016) Adjoint lattice Boltzmann for topology optimization on multi-gpu architecture. Computers and Mathematics With Applications 71(3):833–848, DOI 10.1016/j.camwa.2015.12.043

Zhou BY, Albring T, Gauger NR, Illario da Silva CR, Economon TD, Alonso JJ (2017) A discrete adjoint approach for jet-flap interaction noise reduction. In: Proceedings of 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum, AIAA 2017-0130

Zhu J, Ma J (2013) An improved gray lattice Boltzmann model for simulating fluid flow in multi-scale porous media. Advances in Water Resources 56:61–76, DOI 10.1016/j.advwatres.2013.03.001

Zou Q, He X (1997) On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. Physics of Fluids 9(6):1591–1598, DOI 10.1063/1.869307

Özkaya E, Hay JA, Gauger NR, Schönwald N, Thiele F (2016) A two-level approach for design optimization of acoustic liners. In: Proceedings of 9th International Conference on Computational Fluid Dynamics, ICCFD9-2016-184